2.0 CONCEPTUAL MODEL SPECIFICATION

The Conceptual Model Specification (CMS) presented here has been developed from existing Brawler documentation and software and is intended to substitute for the Software Design Document (SDD) called for in DOD-STD-2167A. Brawler is a mature model that was originally developed during the 1970's, before these standards were in place, and therefore no SDD exists for it. This CMS provides a high level design description for Brawler, along with detailed design descriptions and requirement specifications for selected model components.

This CMS is being developed in stages, and is not yet complete. The current stage includes operational concept and high-level design information. Brawler has been divided into functional areas (top-level functions) which have been further divided into functional elements (FEs) that represent individual model subsystems or functions of the physical systems modeled. The functional areas and elements have been combined into a functional area template (FAT) to allow for consistent numbering and cross-referencing of functional elements across SMART Project documents. The Brawler FAT is presented in Appendix B. Detailed design information for selected functional elements will be incorporated as changes to this document.

2.0.1 OPERATIONAL CONCEPT

Brawler provides a detailed representation of air-to-air combat involving multiple flights of aircraft in both the visual and beyond-visual-range (BVR) arenas. In such engagements cooperative tactics and human factors such as surprise, confusion, and limited situation awareness play a critical role in determining combat outcomes. Accordingly, special emphasis has been placed on carefully simulating these aspects of the engagement process. In addition, a high level of detail is achieved in the hardware models, including those of aircraft aerodynamics, missiles, guns, expendables, radars, missile launch warning devices, radar warning receivers, IRST, IFF, and NCID. Electronic countermeasures versus radars, missiles, and communications are also handled. Hardware models are largely data driven, and data bases describing most current generation US and threat systems for air-to-air engagements are available.

Brawler is used in much the same way as one would use a combat exercise or flight test. Each run simulates a single engagement between multiple aircraft. Just as a succession of trials in a field exercise will produce very different outcomes, repeated runs of Brawler (using different random number sequences), may produce very different engagements. Thus, users typically generate multiple replications of a mission in order to fully explore the set of possible outcomes. Analysis can be facilitated by interactive graphics that permit displays of engagements from many perspectives, and by a report generator that produces statistical analyses of ensembles of engagements. In cases where it has been possible to compare Brawler with field exercises or man-in-the-loop simulations, the statistical distribution of outcomes produced by the model have been quite close to those produced using human pilots.

2.0.2 OVERALL CAPABILITIES

Engagement Size

Brawler is typically used to simulate engagements ranging in size from 2 to 20 aircraft. Larger scenarios can be simulated, with a corresponding run time penalty, by changing a small number of system parameters and recompiling the model.

Aircraft Aerodynamics Model

The aircraft aerodynamics model is a five degree-of-freedom (coordinated flight) model. By five degrees-of-freedom, we mean three translational degrees-of-freedom plus rotation and pitch. Yaw angle is assumed to be zero with respect to the direction of flight. The term coordinated flight means that the use of rudder control is not explicit; any rudder required to appropriately balance yaw forces is assumed.

- Each airframe is described by input data, the bulk of which consists of aerodynamic tables (e.g., coefficient of drag versus coefficient of lift and Mach Number).

- Other airframe data includes radio frequency (RF) and infrared (IR) signature information.

- Munitions and other stores are described separately and are assumed to have add-on contributions to drag and weight.

- Careful attention is paid to accurate lift, angle of attack, fuel flow, and thrust values.

- Maneuver capabilities are modeled as first-order control systems (exponential relaxation to desired roll and pitch rates) with dynamically varying time constants. Many limits to the achievable rates are modeled.

Weapons Models

- Brawler has the capability to simulate five generic weapon types:
  Missiles with multiple seekers IR heat-seeking missiles
  Semi-active radar missiles
  Active radar missiles
  Anti-Radiation Missiles (ARM)
  Guns

- Missile models are complex, although the level at which the aerodynamic properties are modeled is somewhat below that used in the aircraft model.

- Missiles fly according to appropriate guidance laws, react to target aircraft maneuvers, and respond to countermeasures in both their guidance and their endgame calculations.

- Each missile is described by input data that specify its physical characteristics, endgame effectiveness (detailed probability-of-kill tables are accommodated), and guidance parameters.

Radars Modeled

Brawler simulates the operation of pulse Doppler airborne intercept radars with a variety of capabilities and options.

- Radars can operate in medium, high, or interleaved medium/high pulse repetition frequency (PRF) modes.
- Detections are generated by Monte Carlo draws against a probability of detection which is calculated from the signal-to-noise ratio.
- Numerous factors that affect the performance of real radar systems are treated in the radar models, including clutter and ECM (noise jamming and certain deceptive jamming techniques).

Infrared Search and Track (IRST)

The IRST model simulates an IRST characterized by variable scan patterns, an internal trackbank, and an optional laser ranging capability.

- The tracking algorithms attempt to perform self-triangulation when only directional information is available.
- The track bank is updated as the IRST sweeps across a target, so its information is always "current" and ready to be utilized by code that simulates a sensor fusion device.
- IR signatures are modeled at the level of treating plume, skin heating, and engine hot parts separately, and distinguishing between sky and ground backgrounds.
- The directional dependence of signatures is captured.
- The impact of IR clutter is treated statistically, assuming sensors perform spatial and/or temporal filtering.
- Atmospheric transmission is handled for each of several defined IR bands, with an implicit assumption of spectral uniformity within each band.

Radar Warning Receiver (RWR)

The RWR device detects radar emissions from other aircraft and from active missiles, interprets these detections to form a "track" for the entity detected, and conveys the track information to the pilot by means of a display. Both mainlobe and sidelobe emissions are detectable.

The model characteristics include:

- Consideration of frequencies that can be detected
- Field-of-view
- Dwell time in a given direction
- Time required to search the entire FOV
- Signal thresholds

- Maximum track capability and priority rules for dropping tracks when the maximum capability is exceeded.
- Finite resolution.
- The ability to cue a missile approach warning (MAW) device.
- An optional ability to measure range as well as direction is available.
- An optional ability to deduce an emission source for an emission type.

The Brawler pilot interacts with the RWR by looking at the RWR display. The priority he attaches to this interaction depends on the side and number of entities in the RWR sector. Detection of a previously unknown hostile aircraft or missile gives increased value to observing the sector.

IFF/NCID

Identification friend-or-foe (IFF) and non-cooperative identification (NCID) techniques are both treated. Aircraft can be equipped with devices that, when used, permit the determination of the type of another aircraft. These are broadly classed as cooperative (a transceiver is required on the target) or non-cooperative.

- IFF/NCID devices are characterized by reliabilities and characteristic maximum ranges.
- Characteristic ranges can be either single numbers or can reflect interrogator emission power, transponder gain patterns, and a parametric "ECM level."
- Either a generic model, which treats the success of an interrogation probabilistically, or a specific model, which additionally ties the device to the aircraft's radar, may be specified. The capability to utilize hostile IFF capabilities parasitically is also available.

Missile Launch Warning Device

The missile warning device is an IR sensor that detects missiles in their burn phase. The characteristics of the device include:

- A maximum detection range
- A frame, or cycle time associated with the time required to completely look at all areas within the field-of-view (FOV)of the device
- A detailed FOV specification
- Information on the accuracy of directional measurements

The missile warning device triggers the Brawler pilot to look at the device display. He then obtains information from the device track bank on the threat missile.

Expendables

Brawler possesses a model of the effects of expendables on the performance of missiles. The overall level of detail is as follows:

- Expendables may have one of several effects on a missile.
  - The missile may be made to go ballistic shortly after employment.
  - The missile may be made to fly out normally but have the endgame Pk degraded to reflect the effects of the expendable on the missile guidance.
  - The missile may be made to guide on the signal “centroid” of the expendable and the target aircraft.
  - A given expendable type will have only one kind of effect.
- The trajectory of expendables after they are launched is modeled explicitly. Trajectories currently available include:
  - Freely-falling (exponential decay to a vertical terminal velocity).
  - Constant range, constant elevation tow behind the launching aircraft (towed decoy).
- Expendables may be launched by pilots or by missile warning devices. The latter may launch an expendable regardless of whether the pilot is aware of the missile.

Sensor Fusion Devices (SFD) and Situation Awareness Networks (SAN)

Brawler is equipped with a generic sensor fusion device model that integrates detections from designated contributing sensors into a common trackbank.

- Modified Kalman filtering schemes are used to integrate various detections, but correlation problems in real SFDs are not currently treated.
- The impact of SFD is to permit the pilot to more efficiently obtain all information regarding a target, reducing the time required to absorb sensor information and also permitting the pilot to spend more time looking out-of-cockpit.

A situation awareness network model is also available that permits the integration of sensor information from multiple platforms.

- The ability to simulate unreliable or jammed links is present
- The designation of which platforms participate in each SAN is data-driven.
- Ground and airborne intercept and warning entities may be members of a SAN.

Additional Physical Features

Additional physical features of Brawler include:

- The ability to simulate ECM effects on missiles and voice communications.
- Weather (as it affects pilot visual capabilities and IR seekers).

- Ground Controlled Intercept/Airborne Warning and Control System (GCI/ AWACS).
- A surface-to-air missile (SAM) model and a surface-to-surface missile (SSM) model are also available.

2.0.3 TYPICAL USES

Typical applications include hardware design trade-off studies for airframes, avionics and weapons systems, and tactics development. The emphasis on hardware effectiveness is in a realistic mission context. For example, a proposed radar system might have an increased field-of-regard. Brawler could determine the degree to which this improvement enhances the ability to perform a specific mission. Brawler has also been embedded as an intelligent target generator in several manned simulator facilities, where it enhances the ability to simulate large engagements.

2.0.4 ASSUMPTIONS AND LIMITATIONS

Top Level Software Design

Brawler is an event driven simulation. At the highest level, the major components consist of the event scheduling and execution software and a separate component for each event type. Event types generally correspond to either physical processes, such as weapon flyouts, that are simulated by Brawler, or simulation management events that relate to the structure and execution of the simulation. The various events are described below.

Overview of Physical Process Events

This section presents an overview of the different event types implemented in version V6.2 of Brawler that simulate hardware functions or human behavior. Many of these events are not scheduled at regular intervals, but are timed to match the activity of the systems they represent. For example, radar frame events are scheduled according to the frame time of the radar, and sweep events are scheduled to occur at the time when the radar actually sweeps the target.

Physical process events that occur repeatedly, such as consciousness events, are self-planting. This means that when the event finishes execution, it schedules its own next event. This allows the timing of events to change dynamically to reflect status changes such as sensor mode changes. Events also check for their own obsolescence. For example, when a consciousness event executes, it checks to see if the pilot is still alive. If not, the event terminates and does not plant another consciousness event for that pilot. A separate stream of events is generated for each event type that is associated with a single entity. For example, there are separate radar events for every radar in the simulation.

Physical Process Event Types

<u>Consciousness Event</u>

This event simulates the decision making pilots and SAM site operators. A consciousness event can be broken up into several phases, which are executed sequentially.

Situation Update

During the situation assessment phase, the pilot makes new observations of other entities, either visually, via receipt of a radio message, or by observing a sensor display. All of these processes are simulated. The form of an observation differs for different types of sensors and sensor modes. New observations are then used to update the pilot's perception, or "mental model" of the rest of the engagement.

Situation Assessment

The pilot's assessment of the current situation involves the evaluation of many derived variables that define its various aspects. For example, functions are evaluated that reflect the assessed intent of other aircraft, the degree to which one aircraft is threatened by another, and the probability that an aircraft has been detected by a hostile flight. These variables are generally expressed as either "surrogate probabilities" or as expected aircraft values. A surrogate probability is a number in the range 0-1 that may be thought of loosely as the probability that an event will occur.

Decision Making

During this phase, the pilot uses his current situation perception to make decisions regarding tactics, maneuvers, sensor management, weapon selection and firing, and whether or not to send radio messages. Once the pilot has made a decision, the actions that he will take to implement the decision are simulated.

<u>Communications Routing</u>

The communications routine event executive is used to determine when and if radio messages can be transmitted and takes into account the current channel usage and the presence of communications jamming into account.

<u>Weapon</u>

Missile and gun event divided into the following event subtypes:

Internal Launch - Handles the launch of missiles by aircraft controlled by Brawler.

Missile Fly - Handles the flyout of missiles.

Missile Endgame - Executed when a missile fuzes, this function computes the result of the missile detonation. If successful, it also kills the target aircraft and performs any associated statistics collection and data structure cleanup.

Removal - Handles the removal of missile entities from the simulation after they have detonated or gone ballistic.

Gun Endgame - Executed after a successful gun shot to kill the target aircraft and perform any associated statistics collection and data structure cleanup.

External Launch - Handles the launch of missiles that are fired by another simulation but are to be flown out. This allows other simulations in a model confederation to retain their own weapon firing logic while using the Brawler missile model.

Rail - Handles simulation of seeker functioning for a missile before it leaves the launching aircraft. This is done so that the effects of various countermeasures on seeker acquisition can be simulated pre-launch.

Stand-off Jammer Update

Stand-off jammer orbits are specified as a series of straight, constant speed segments. Each time a jammer reaches the end of the current segment, an update event is executed to reset the speed and direction data describing the current segment to the next segment.

Communications Jammer Update

Resets the parametric jamming level for each communications channel based upon current channel usage. In other words, the more heavily a channel is being used, the more heavily it will be jammed, forcing users of that channel to change channels or lose the ability to communicate. These events are executed regularly, at an interval that is set by the user in the scenario file.

GCI/AWACS Controller Consciousness

This event simulates the decisions made by GCI/AWACS controllers. It has a similar structure to the pilot consciousness event, in that it begins with a situation update phase, then has a situation assessment phase, and finishes with a decision phase and action phase. It differs in that the decisions and actions are all concerned with what messages to transmit to aircraft under the direction of the controller. Messages can be simple observations, flight vectors, or directions to execute specific tactics.

IRST Frame

Performs geometry and timing calculations to determine what targets might be detected during the next IRST frame. For each potential detection, an IRST detection event (described below), is planted.

IRST Detection

This event simulates a single sweep of an IRST across a target. It computes the received signal and determines whether or not a detection is made. If so, the IRST trackbank is updated for that target.

Jammer Status Control

Jammers can be off, on, or operating in a responsive mode, where they are off until swept by a threat radar, at which time they turn on until the radar stops sweeping them. Jammer status control events simulate this responsive behavior by turning jammers on and off based upon the time and geometry at which each threat radar last illuminated the jammer.

Missile Warning

Missile launch warning (MW) events are planted when a missile is launched. A separate event is planted for each target that might be able to detect the missile. When the event executes, it determines whether or not a detection is made. If a detection occurs, an observation is generated and an alarm is triggered which causes the immediate execution of a pilot consciousness event. Finally, an event is planted to cue the missile approach warning radar, if the aircraft has one, and another MW event is planted for 0.1 seconds in the future.

<u>Radar</u>

This is the event executive for the radar model. It is broken into the following event subtypes:

> *Frame* - Determines current radar mode and pattern. Then, determines what targets may be detected during the next frame. For each of these, a radar sweep event is scheduled.
>
> *Sweep* - Handles a single sweep of the radar across a target (or group of non-resolvable targets). Determines whether or not a detection occurs, taking into account target signal, noise, jamming, and clutter. If a detection occurs, this event also generates an observation of the target.
>
> *Miss* - The radar model determines the status of tracks (established vs. not established) based upon the criterion of *m* detections during the previous *n* frames, where m and n are input data elements. Miss events are scheduled to track frames during which no detections occur.
>
> *Bar* - Brawler maintains an RF blackboard that records the RF emissions produced by active sensors during an engagement. Radar bar events are executed to post the emissions produced by each bar in a radar frame as the bar is swept. Emissions for all of the bars in a frame are not posted at the beginning of the frame because the radar may change mode or pattern before the frame finishes.
>
> *Switch Change* - Brawler pilots can change the mode, pattern, or pattern position of the radar. Switch change events are planted to simulate the time delay between the pilot's decision to change a radar setting and the response of the radar.

<u>Radar Warning Receiver</u>

Radar warning receiver (RWR) events are executed at regular intervals to determine whether RWR devices can detect sidelobe emissions from radars carried on other platforms. (Detections of mainlobe emissions are also calculated, but these calculations are done at the time the emitter illuminates the RWR.) RWR sidelobe events traverse the RF blackboard and evaluate candidate emissions that are posted there. Tests on emission type, timing, power level, and location are performed.

<u>Expendables</u>

This event type handles the simulation of expendables and is broken into the following subtypes:

> *Creation* - Handles the initialization of expendable entities after they have been launched.

Projection - Handles the physical projection of expendables after they have been launched.

Removal - This event subtype handles the removal of expendables from the simulation.

Mode Change - Users can change the mode of a repeater type expendable after it has been launched via a production rules utility. This event subtype implements the mode change.

Command Guided Missile Update

Handles the transfer of command guidance data from the aircraft transmitting the updates to the missile. These events execute at an interval specified in the guidance data for the missile.

Missile Approach Warning

Event executive for the missile approach warning (MAW) radar. Determines whether or not detections occur for missiles already being tracked as well as in response to cueing from other avionics devices such as a missile warning device. Updates a trackbank in response to new observations. The MAW is also capable of launching expendables.

Situation Awareness Network

Brawler simulates situation awareness networks (SANs) using a central trackbank that is fed by observations from all of the platforms on the net. Each time an observation is sent to the net, a SAN event is scheduled to create or update the correct track.

IFF event

Computes the result of a pilot employing an IFF device to type an unknown aircraft. A determination, based upon geometry, device types, and probabilities based upon device reliability and interference, is made as to whether or not the attempt succeeds. If so, the type of aircraft will be displayed on the pilot’s radar or sensor fusion device scope.

Overview of Simulation Management Events

Brawler also executes a number of event types that are not directly related to physical processes. These events perform activities such as reporting data from the simulation for later analysis, changing diagnostic prints at times specified by the user, internal data structure cleanup, or controlling the execution of the simulation when running in confederation with other simulations.

Simulation Management Event Types

Diagnostic Print Modifications

Brawler maintains a large set of diagnostic print switches that may be used by the analyst to report detailed information from different components of the model during execution. These can be useful for validation, debugging, study definition, or statistics collection. The initial state of the diagnostic print switches is read from the scenario file during initialization. The user also has the option of changing the state of the print switches during an engagement to reduce the amount of output data produced. Time dependent print modifications are also specified in the scenario file by planting a print modification event. When the event is executed, the status of the print switches is changed to the settings specified by the user.

History File Output

During the course of an engagement, information is written to a history file. The history file can then be used by the graphics post-processor and the event summary utility program. Data related to non-periodic events, such as weapon firings, is written to the history file as the events are executed. In addition, periodic updates of the current state of all players in an engagement are also written. These periodic updates are performed during the history file output events. The frequency of history file events is set in the scenario file by the user and is read during initialization.

Checkpoint

The simulation can be restarted from a checkpoint file, which allows the user to pick up execution in the middle of an engagement. This can be useful when debugging problems that occur late in a complex scenario. Checkpoint files are generated during checkpoint events. During a checkpoint event, all of the data needed to restart the engagement is written to the checkpoint file. The times at which checkpoints are to be taken are set by the user in the scenario file.

RF Blackboard Cleanup

RF blackboard events are executed periodically to remove obsolete entries on the RF blackboard.

Confederation Synchronization

When Brawler is being run in a non-real time confederated mode with another simulation, synchronization events are executed at regular intervals to exchange data between the simulations. During a synchronization event, incoming messages are read and processed, messages about current Brawler players are transmitted, and any aircraft that are to be transferred to the other simulation are handed off. The event ends by sending an “end of data” message that signals the other simulation that transmission is finished. The synchronization interval, typically on the order of one second, is a user defined parameter.

2.0.5 LOGIC FLOW THROUGH MAJOR COMPONENTS

The flow of logic during Brawler execution is shown in Appendix A. The upper left corner of the figure is the start of program execution after all initialization activities have been completed. This is indicated by an “A” enclosed by a circle. All other enclosed As indicate a return to this point. Similarly, other enclosed letters indicate transitions from one part of the diagram to another. An enclosed Z indicates termination of the simulation.

The initial portion of the diagram shows the event scheduling and execution of the main simulation. The logic flow through the consciousness, radar, and weapon events is expanded in subsequent sections. Internal and external missile launch events are presented under the weapon events as a single event type because they do not differ significantly in their logical organization. The principal differences between them are in the set of data structures that have to be initialized during the event. Circles containing the word "End" in the event specific sections indicate the end of event execution and a return to the first page of the diagram.

2.0.6 DATA FLOW THROUGH MAJOR COMPONENTS

At the highest level, data flow through Brawler is depicted in Figure 2.0-1. The ovals represent various event types and the rectangles represent various types of data. The "Central Status" data structures contain the ground truth states (position, velocity, dead/alive, etc.) of all of the entities in the simulation. "Characteristics Data" are the data that describe the physical characteristics of each type of hardware system in the simulation. Characteristics data are read from input data files and do not change during the course the simulation. "Status Data" are data that describe the current state of each system or subsystem (radar power, mode, number of tracks, etc.) in the simulation. Status data will change during the course of the simulation. The "Mental Status Arrays" or "mental model", contains a pilot or other operator's current perception of the situation.

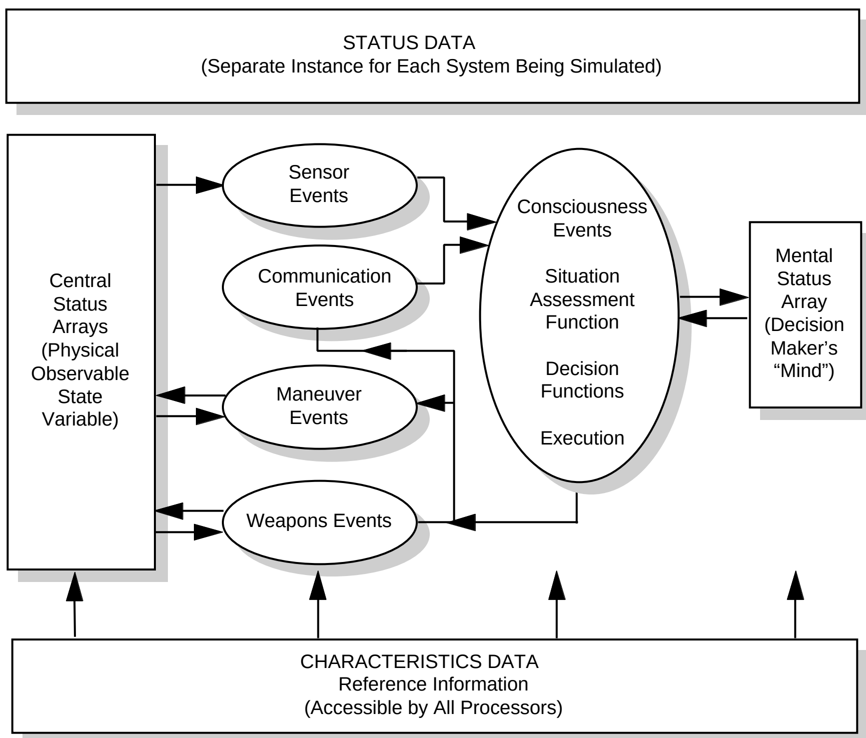

FIGURE 2.0-1. Brawler Data Flow.

Each hardware system has associated characteristics and status commons to hold the data associated with it. Because characteristics do not change, there need only be one copy of the characteristics data for a given type of system. Because status data will change, there will be a separate set of status data for each instance of each type of system. For example, a scenario that runs three radars of type A and four of type B will maintain one copy of type A characteristics, one copy of type B characteristics, three copies of type A status and four of type B status. Each time an event for a particular hardware system is executed, it will make use of the characteristics and status data for that system.

Likewise, each conscious player has a separate mental model. At any time, the contents of each pilot's mental model will be a function of the observations made by that pilot up to that point. Pilot decision making is based upon the contents of the mental model, *not* upon ground truth. This means that the pilot's decision making is sensitive to the timeliness and accuracy of the information provided by sensors and visual observations.

The basic flow of information through Brawler, then, is that the sensor models use ground truth data, sensor characteristics, and status to produce detections. When a pilot looks at a sensor display, those detections are observed and are incorporated into the pilot's mental model. Decisions use the contents of the mental model to produce actions, which then cause changes in hardware status and ground truth by maneuvering aircraft, changing avionics modes, sending messages, and launching missiles and expendables.

Input data consists of a scenario file that describes the initial loadouts, organization, and disposition of the players in the simulation, data files that contain the characteristics data for all of the systems to be used, and optional production rules data files that contain inputs to user-written extensions to the simulation. Every event executed can write event related data to the measures of performance (MOP) database, the diagnostic output file, **IOUT**, or to the terminal screen. Periodic updates of entity state vectors and information related to pilot decision making weapons and expendables employment are also written to the history file, **HIST**, for later use by graphical post-processors.

2.0.7 SOURCE CODE HIERARCHY

The top-level source code hierarchy for Brawler is presented in Figure 2.0-2. A brief overview of the functions of the subroutines in the hierarchy is presented here. This is followed by a more detailed expansion of three representative event executives.

Brawler is an event-driven simulation, and the top level subroutines are the ones associated with scheduling and execution of events. As is shown in the figure, subroutines *prgini* and *tflite* are initialization routines, dealing with program initialization and reading from the various input data files. Subroutines *simdrv*, *donext*, and *dnxtev* handle the scheduling and execution of events. Subroutine *finish* is called at the end of the simulation to close data files, perform final statistic computations, and to print any final diagnostics.

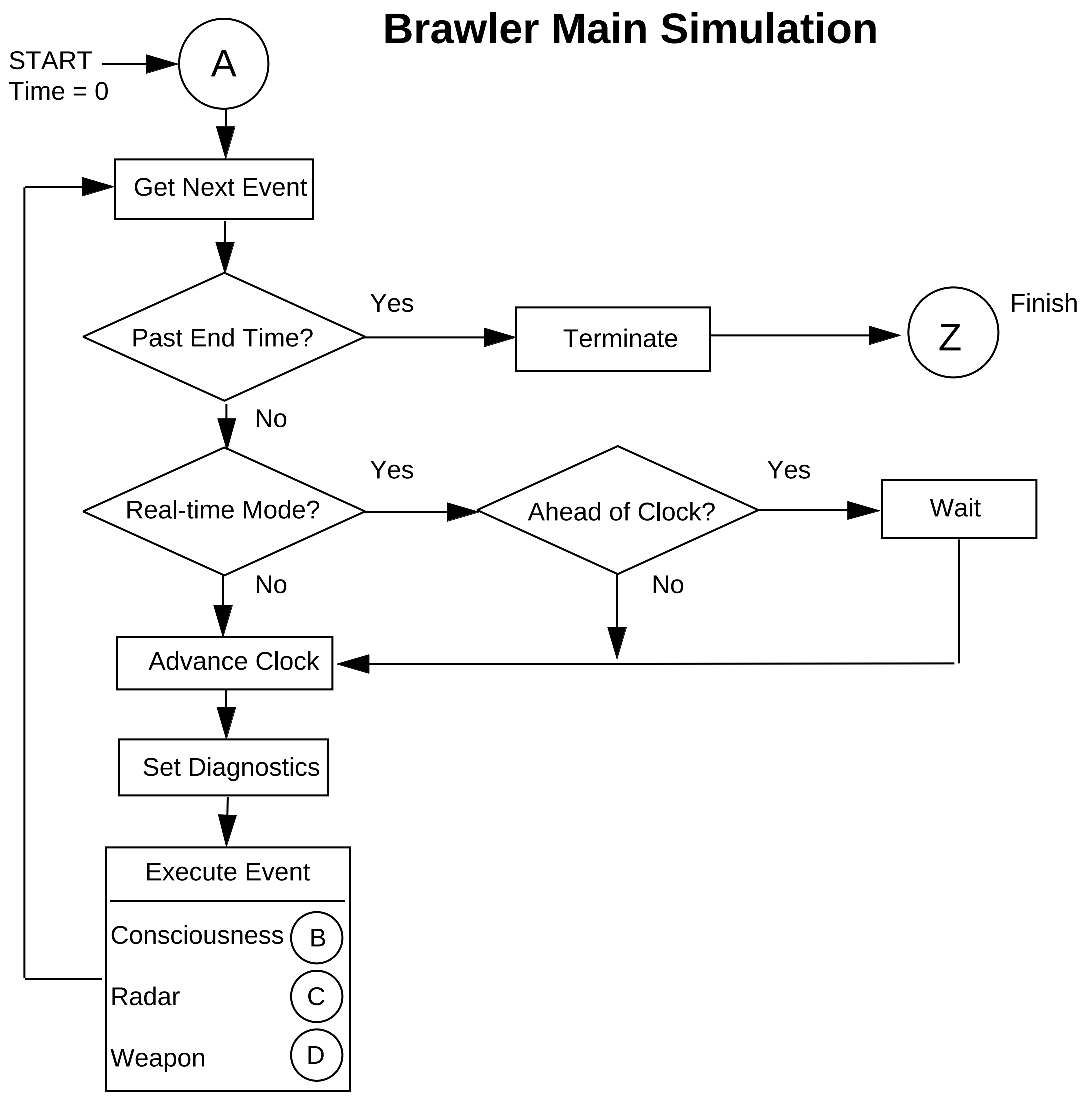

FIGURE 2.0-2. Top Level Source Code Hierarchy.

Below *dnxtev* are event executives, each of which handles the simulation of one of the event types modeled. The figure lists the executives for the pilot consciousness, weapon, and radar event executives. Brief descriptions of the various event types are given above. Table 2.0-1 lists the names of the event executives.

TABLE 2.0-1. Brawler Event Executives.

Event Executive	Event
	Physical Process Events
conevt	Consciousness
comrte	Communications routing
mslevt	Weapon
sojevt	Stand-off jammer update
cjamev	Communications jamming strength update
gcievt	AWACS/GCI controller consciousness
framev	IRST frame
irstev	IRST detection
jcevt	Jammer status control
mwevt	Missile launch warning (MW)
rdrevt	Radar
rwrevt	Radar warning receiver (RWR) sidelobe detection
expevt	Expendable
updtev	Missile command guidance update
mawevt	Missile approach warning (MAW)
sanevt	Situation awareness net (SAN)
iffevt	IFF device
	Simulation Management Events
prnevt	Diagnostic print switch modification
hevt	History file update
chekpt	Checkpoint
rfclev	RF blackboard cleanup
sync_evt	Confederation synchronization and data exchange event

Consciousness event source code hierarchy

Figure 2.0-3 is an expansion of the consciousness event source code, showing the major components of that event.

Subroutines *flyac* and *perfrm* are used to bring the aircraft of the currently conscious pilot up to the current simulation time and to update the pilot's knowledge of the current performance capabilities of the aircraft.

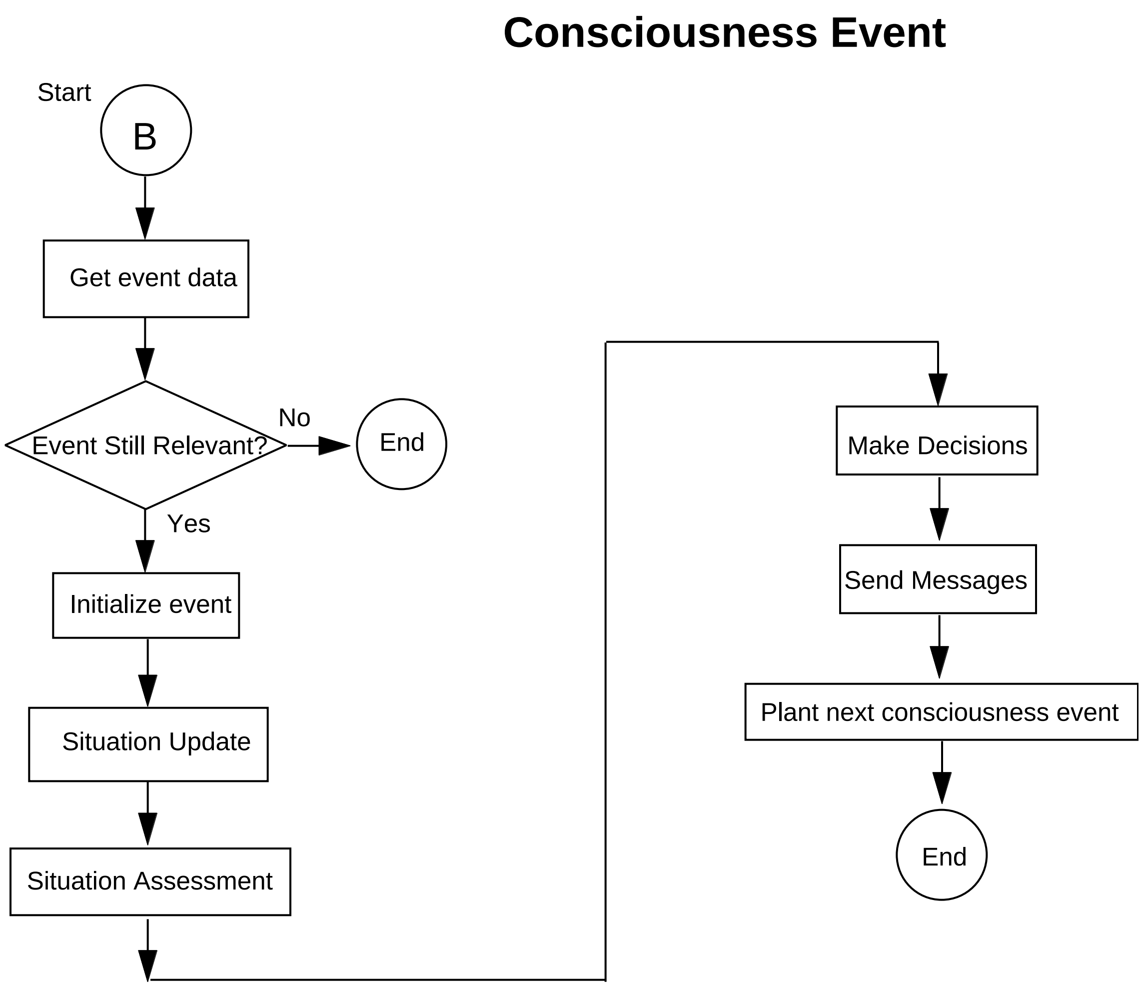

FIGURE 2.0-3. Consciousness Event Source Code Hierarchy.

Subroutine *sensor* simulates the pilot's observations of the sky and various cockpit avionics displays. Under *sensor*, subroutine *setsvl* ("set sector values") determines the order in which the pilot will look at various sky sectors and avionics displays. Subroutine *sctsch* ("sector search") models the visual observation of each sector in the computed order until time runs out. The routines under *sctsch* each simulate the observation of one visual sector, computing the time required to observe that sector as well as the observations that are actually made.

Subroutine *inferl* ("infer, late", late because the inference occurs after the observations have been processed) simulates the inferred detection of previously unknown aircraft that may result from observations made under sensor. For example, observation of a missile implies the existence of a launcher. If the pilot is not aware of any aircraft in the vicinity of the missile launch, he is allowed to infer the presence of the launcher without having directly observed it.

Subroutine *mindup* ("mind update") simulates the incorporation and evaluation of new information into the pilot's situation perception, or "mental model." Under *mindup*, subroutine *rcvobs* ("receive observations") incorporates new observations received via radio messages from other aircraft or GCI/AWACS controllers. Subroutine *cc2x0* handles the update of pilot awareness of the physical state (position, velocity, type, side, etc.) of all of the players in the engagement of which the pilot is aware. A similar routine, *msl2x0*, not pictured, performs a similar function for observations of missiles.

Subroutine *astyp* ("assign type") is used to try to assign a type to previously untyped aircraft. Brawler pilots are allowed to infer the types of other aircraft if they are flying in formation with aircraft whose types are known or if they exhibit overtly hostile actions such as firing upon aircraft known to be friendly. Related routines *asown* ("assign missile owner") and *astgt* ("assign missile target"), not shown, are called to try to assign owners and targets to missiles based upon observed missile trajectory and the current states of other aircraft.

Subroutine *rcv_intent* ("receive intent to fire") is called to process the receipt of an "intent to fire" message from a friendly aircraft. These messages are sent to prevent two or more aircraft from mistakenly targeting the same hostile.

It is possible to limit the number of other players that a pilot can actively consider when making decisions. This number can be a global limit on all pilots or it can be a function of pilot skill level. It can also vary with the stress level of the simulated pilot. Subroutine *mmordr* ("mental model ordering") is called to prioritize the aircraft in the pilot's mental model and then divide them into a "detailed consideration group" and all other players. The assigned priorities result from a combination of factors, including relationship to the pilot (i.e., flight leader gets a higher priority than another wingman), relative geometries, and optional user input.

If the new observations made during this consciousness event contain significant information, such as detections of new players, subroutine *majud* ("major update") is called to update the pilot's assessment of the current situation. During a major update, situation assessment measures are calculated that will be used later during the decision making process. Situation assessment measures include factors such as lists of which players are threats, force ratios, probabilities of kill and survival, etc. If the new observations do not include significant information, subroutine *minud* ("minor update") is called to perform a less extensive minor update.

Subroutine *preobs* ("prepare observation messages") is called to construct radio messages from the currently conscious pilot to other players on his channel regarding newly detected or typed aircraft or aircraft kills that were just observed.

After the mental model update has been completed, *modsel* ("mode selection") is called to simulate pilot decision making. Under *modsel*, *pcode* ("production rules code") is the user interface to the decision making process. Through these production rules, the user can issue instructions or change the values assigned in mindup to reflect specific mission objectives. Subroutine *setspt* ("set successor pointers") determines which alternative courses of action are to be considered for each kind of decision to be made. Subroutine *declev* ("decision level") determines what type of decision (flight posture, flight tactics, pilot posture, maneuver, etc.) is to be made. Subroutine *pkactn* handles the generation, projection, and scoring of available alternatives for the selected decision type, as indicated by the lower level routines *aslctX*, *aprojX*, and *aevalX*. The results of the decision are implemented in one of the *akshnX* routines.

Subroutine *fncom* ("final communications") is called to send any messages that may have been prepared during this consciousness event. This can include observation messages, requests for information, intent to fire, or targeting assignments if the conscious pilot is a flight or element leader.

Subroutines *gcetim* ("get consciousness event time") and *makece* ("make consciousness event") are called to schedule the next consciousness event for this pilot. *gcetim* determines when the event is to occur and *makece* fills in the data structures and plants the event in the event heap.

Weapon event source code hierarchy

Figure 2.0-4 is an expansion of the weapon event source code, showing the major components of that event. The missile event is broken into several subtypes, each of which is discussed below.

Subroutine *rail_ev* handles the modeling of missiles after they have been selected but before they have been fired. During this interval, the missile seeker and trackbank code are executed so as to make the missile model sensitive to countermeasures that affect seeker acquisition and, hence, delay firing. Subroutine *all_msl_skrs* simulates the functions of all missile seekers that are on before the missile is fired. This includes simulation of seeker movement and detection of signals. Subroutine *msl_tkb_upd* models the incorporation of seeker observations into the missile's internal trackbanks. Subroutine *cage_skr* recages the missile seeker if it loses track of its target either due to hitting the seeker gimbal limit or due to loss of signal. Finally, *rail_acq* determines whether or not the missile has acquired a target. If more than one possible target is being tracked, the missile will acquire the strongest signal, but only if it exceeds the others by a user specified amount. *rail_acq* is structured so that it can be extended to accept other acquisition algorithms.

Subroutine *mslaun* simulates the actual launch of a missile. During a launch event, subroutine *mslrls* is called to determine whether or not the missile successfully separates from the aircraft. If so, the drag of the launcher is adjusted, the initial physical state of the missile is set up, initial command guidance data is transmitted, if appropriate, and a final rail event is executed to bring the seekers and trackbanks up to date for any seekers on at launch.

Subroutine *mslaun_e* is executed for "external" missile launches. An external launch is one that occurs when Brawler is running in confederation with another simulation and that other simulation fires a missile that Brawler is to fly out. In such a case, *mslaun_e* is called to initialize the physical state of the missile at the moment of launch and to begin the series of flyout events for the missile.

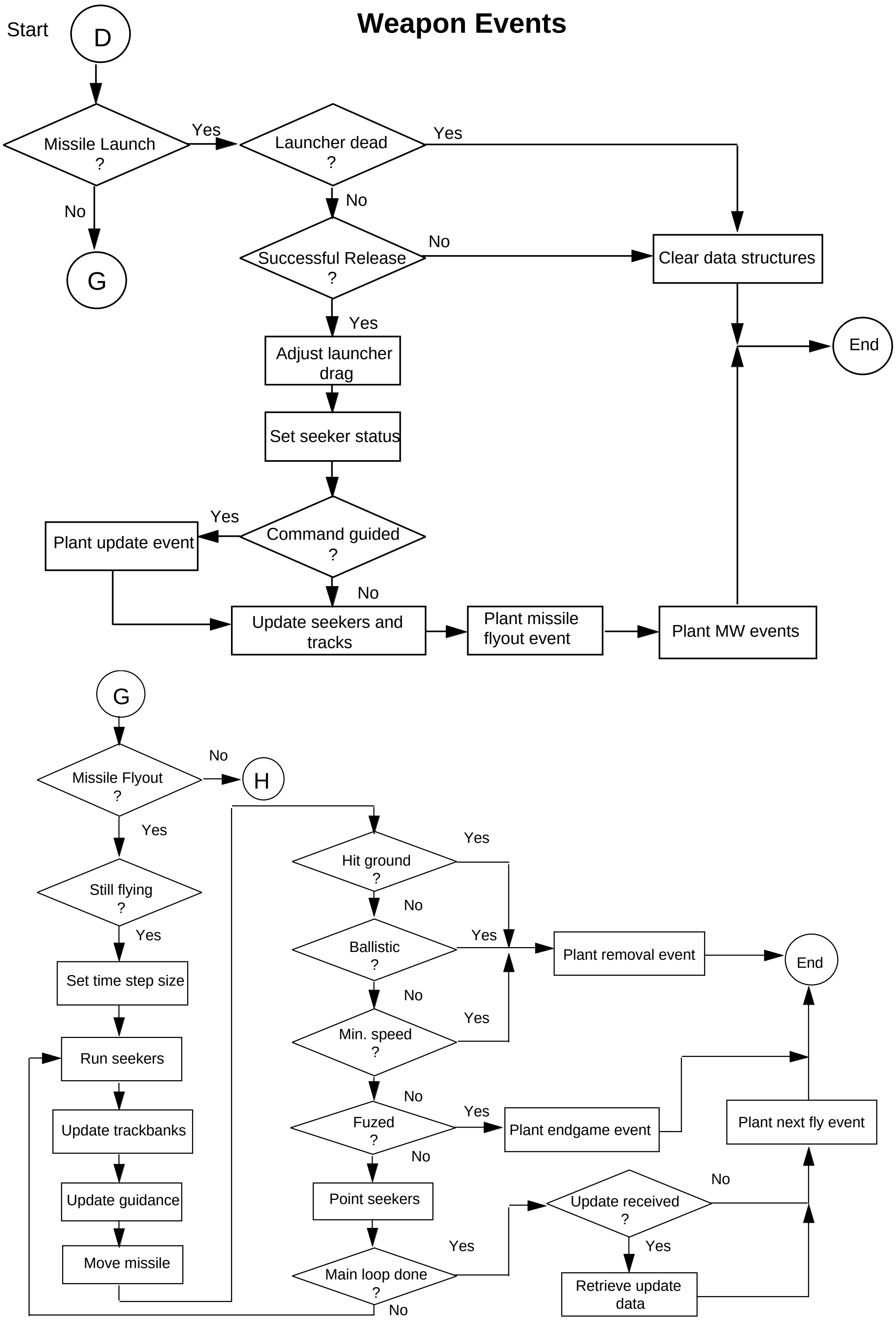

FIGURE 2.0-4. Weapon Event Source Code Hierarchy.

Subroutine *mslfev* is the missile flyout event. It is executed at approximately one second intervals for each missile in the air. During a fly event, subroutine *mslfly* is called to simulate the missile's activities from the end of the last fly event for this missile up to the current simulation time. *mslfly* generally divides this time interval into smaller time steps and executes each of the missile functions described below once during each time step. This is done to improve the accuracy of the simulation. *mslfly* executes the same calls to *all_msl_skrs* and *msl_tkb_upd* to determine missile seeker observations and trackbank updates. Subroutine *mcntrl* selects a source for guidance inputs, either command guidance data or an internally maintained target track, then calls one of several guidance routines to execute the guidance law specific to that missile. Guidance commands from *mcntrl* are fed into *msldyn* to move the missile forward according to the missile's current state and aerodynamic characteristics. Subroutine *mslfuz* determines whether the missile has satisfied its fuzing criteria, and, if so, whether or not fuzing was successful. It should be noted that successful fuzing does not imply a target kill, and that the determination of whether or not the missile kills is deferred until the endgame event is executed. If, during its flyout, the missile fails, either due to countermeasures or one of a number of possible system failures whose probabilities are specified as input data, *mblstc* is called to put the missile into a ballistic state. If the missile is not ballistic and hasn't fuzed, *skrdir* is called after the missile has been moved by *msldyn* to reorient the seeker. Seeker movement is a function of target track position, missile movement, and seeker gimbal and rate limits. After *mslfly* is executed, *updgui* is called to incorporate newly received command guidance information, if any has been received.

The missile endgame event determines the results of a successful missile fuzing. The top level routine in this event is *mslend*. It calls *calc_pk* to compute a probability of kill based upon endgame geometry and the probability of kill algorithm specified for the missile. After this, subroutines *jpkfac* and *expdeg* are called to account for the effects of any jammers or expendables seen by the missile whose effect is to be modeled by degrading the missile's probability of kill. Once an adjusted probability of kill is computed, a random draw against it is made and, if successful, the aircraft is destroyed by a call to *killac*.

Once a missile has either exploded or gone ballistic, a missile removal event is planted to remove the missile from the simulation. The removal occurs immediately for missiles that have exploded. For other missiles, the removal is delayed because pilots may not know the missile has failed and should still have the opportunity to observe and react to it until it has fallen well away from the engagement. Subroutine *mremev* performs the missile removal.

Gun firing is modeled more simply than missiles. No explicit bullet flyout events are executed. Instead, the results of a gun shot are computed at the time that the shot is taken, and if the shot succeeds, a gun endgame event is planted. Subroutine *gendgm* executes this event, and consists of little more than a call to *killac* to destroy the target.

Radar event source code hierarchy

Figure 2.0-5 is an expansion of the radar event source code, showing the major components of that event.

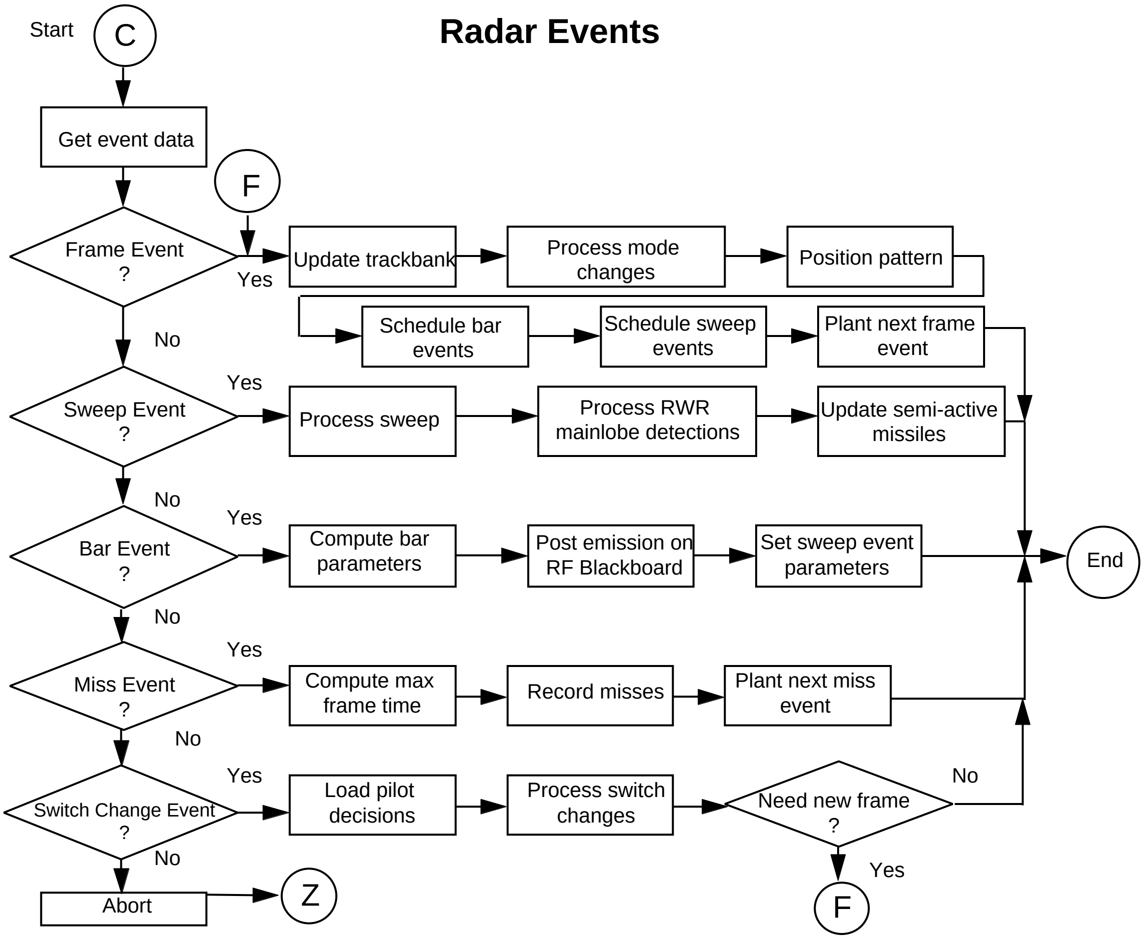

FIGURE 2.0-5. Radar Event Source Code Hierarchy.

As with the weapon event discussed above, the radar event is divided into several subtypes, each of which simulates different radar functions. The first of these is the frame event, whose top level subroutine is *rdrfev*. The frame event simulates the overall functioning of the radar. It is during this event that the current mode is determined, along with the scan or TWS pattern selection and positioning. Scan and TWS patterns consist of a set of scan bars of a set width, scan rate, and angular separation. Once the pattern has been determined, subroutine *schbar* is called to plant bar events for each bar in the pattern. The frame event also calls subroutine *rrinfm* to determine what targets will be illuminated during the current frame, and *rrscsw* to schedule a radar sweep event for each target in the frame.

Subroutine rrbrev implements the bar event. The purpose of a bar event is to determine the radar emission parameters for the bar and to post an emission record on the Brawler RF blackboard that records this information. Bars are not posted during the radar frame event because the radar may change mode during a frame, in which case not all of the bars would be scanned.

Subroutine *rdrsev* is the top level routine for the radar sweep event. A sweep event is executed to simulate a single sweep of the radar across a target or a group of unresolvable targets. During a sweep event, subroutine *rrdtct* is called to determine whether or not a detection occurs and, if so, to compute the observation data and update the radar trackbank. Subroutine *rwr_ml_det* is called to execute RWR mainlobe detection code for each target that has been illuminated that is carrying an RWR device. Subroutine *rrsnm* is called to schedule missile fly events for any semi-active missiles fired at the target that are being supported by this radar.

The radar miss event is executed by subroutine rdrmev. Miss events are used to determine whether or not a radar track has timed out and been disestablished or purged because the radar has not made any recent observations of that target.

Finally, the radar switch change event is executed to process changes in radar mode, pattern, or position that have been dictated by the pilot or the user. Whenever a Brawler pilot decides to change his radar mode or reposition the radar pattern, this decision is recorded as a desired change. A switch change event is then planted for a short time in the future. The time delay accounts for human reflexes and the fact that radars cannot change instantaneously. When the switch change event is executed by a call to *rdr_sw_chg*, the desired radar settings are translated into actual settings and a new radar frame event is executed by a direct call to *rdrevt*.

2.0.8 IMPLICATIONS FOR MODEL USE

This design is focused on sequential processing of events and information perceived via pilot observations rather than upon effectiveness of aircraft and their weapons. Even through relative differences in effectiveness can be inferred from outcomes, proper use of the simulation should be focused on more subtle timing, reactions, and decisions that contribute to those results. In other words intended uses of the software were planned to encompass evaluations of factors that affect pilot perception, reactions, and performance rather than comparisons of aircraft platforms, sensors and weapons.